7-13-00

A

jc877 U.S. PTO
07/12/00

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Please type a plus sign (+) inside this box → [+]

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

Attorney Docket No.	RAVI0009
First Inventor or Application Identifier	Sigmund
Title	On the Fly Generation of Multimedia Code for Image...
Express Mail Label No.	EL540885913US

jc658 U.S. PTO 09/614635 07/12/00

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231

1. [X] * Fee Transmittal Form (e.g., PTO/SB/17) (Submit an original and a duplicate for fee processing)
2. [X] Specification [Total Pages 34] (preferred arrangement set forth below)
   - Descriptive title of the Invention
   - Cross References to Related Applications
   - Statement Regarding Fed sponsored R & D
   - Reference to Microfiche Appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings (if filed)
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
3. [X] Drawing(s) (35 U.S.C. 113) [Total Sheets 1]
4. Oath or Declaration [Total Pages 2]
   a. [X] Newly executed (original or copy)
   b. [ ] Copy from a prior application (37 C.F.R. § 1.63(d)) (for continuation/divisional with Box 16 completed)
      i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

*** NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).**

5. [ ] Microfiche Computer Program (Appendix)
6. Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
   a. [ ] Computer Readable Copy
   b. [ ] Paper Copy (identical to computer copy)
   c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

7. [ ] Assignment Papers (cover sheet & document(s))
8. [ ] 37 C.F.R.§3.73(b) Statement (when there is an assignee) [X] Power of Attorney
9. [ ] English Translation Document (if applicable)
10. [X] Information Disclosure Statement (IDS)/PTO-1449 [X] Copies of IDS Citations
11. [ ] Preliminary Amendment
12. [X] Return Receipt Postcard (MPEP 503) (Should be specifically itemized)
13. [X] * Small Entity Statement(s) (PTO/SB/09-12) [ ] Statement filed in prior application, Status still proper and desired
14. [ ] Certified Copy of Priority Document(s) (if foreign priority is claimed)
15. [ ] Other: ....

16. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment:
[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No ______/______
Prior application information: Examiner ______ Group / Art Unit: ______

For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS

[X] Customer Number or Bar Code Label: 22862 (Insert Customer No or Attach bar code label here) or [ ] Correspondence address below

Name					
Address					
City		State		Zip Code	
Country		Telephone		Fax	

Name (Print/Type)	Michael A. Glenn	Registration No. (Attorney/Agent)	30,176
Signature		Date	7/12/2000

PTO/SB/10 (1-99)
Approved for use through 09/30/2000. OMB 0651-0031
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

STATEMENT CLAIMING SMALL ENTITY STATUS
(37 CFR 1.9(f) & 1.27(c))--SMALL BUSINESS CONCERN

Docket Number (Optional) RAVI0009

Applicant, Patentee, or Identifier: Sigmund
Application or Patent No.:
Filed or Issued: Herewith
Title: On The Fly Generation of Multimedia Code for Image Processing

I hereby state that I am
☐ the owner of the small business concern identified below:
☒ an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF SMALL BUSINESS CONCERN Ravisent Technologies, Inc.

ADDRESS OF SMALL BUSINESS CONCERN 1 Great Valley Parkway, Malvern, PA 19355-1308

I hereby state that the above identified small business concern qualifies as a small business concern as defined in 13 CFR Part 121 for purposes of paying reduced fees to the United States Patent and Trademark Office. Questions related to size standards for a small business concern may be directed to: Small Business Administration, Size Standards Staff, 409 Third Street, SW, Washington, DC 20416.

I hereby state that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the invention described in:

☒ the specification filed herewith with title as listed above.
☐ the application identified above.
☐ the patent identified above.

If the rights held by the above identified small business concern are not exclusive, each individual, concern, or organization having rights in the invention must file separate statements as to their status as small entities, and no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d), or a nonprofit organization under 37 CFR 1.9(e)

Each person, concern, or organization having any rights in the invention is listed below.
☐ no such person, concern, or organization exists.
☒ each such person, concern, or organization is listed below.

Separate statements are required from each named person, concern or organization having rights to the invention stating their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

NAME OF PERSON SIGNING Ned Barlas

TITLE OF PERSON IF OTHER THAN OWNER VP. Chief Legal Officer

ADDRESS OF PERSON SIGNING 1 Great Valley Parkway, Malvern, PA 19355-1308

SIGNATURE Ned E. Barlas DATE March 29, 2000

PTO/SB/09 (12-97)
Approved for use through 9/30/00 OMB 0651-0031
Patent and Trademark Office; U.S DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

STATEMENT CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) & 1.27(b))--INDEPENDENT INVENTOR	Docket Number (Optional) RAVI0009

Applicant, Patentee, or Identifier: Sigmund

Application or Patent No.: Unassigned

Filed or Issued: Herewith

Title: On The Fly Generation of Multimedia Code for Image Processing

As a below named inventor, I hereby state that I qualify as an independent inventor as defined in 37 CFR 1.9(c) for purposes of paying reduced fees to the Patent and Trademark Office described in:

- [X] the specification filed herewith with title as listed above.
- [ ] the application identified above.
- [ ] the patent identified above.

I have not assigned, granted, conveyed, or licensed, and am under no obligation under contract or law to assign, grant, convey, or license, any rights in the invention to any person who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person had made the invention, or to any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

Each person, concern, or organization to which I have assigned, granted, conveyed, or licensed or am under an obligation under contract or law to assign, grant, convey, or license any rights in the invention is listed below.

- [ ] No such person, concern, or organization exists.
- [X] Each such person, concern, or organization is listed below.

Separate statements are required from each named person, concern, or organization having rights to the invention stating their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

ULRICH SIGMUND		
NAME OF INVENTOR	NAME OF INVENTOR	NAME OF INVENTOR
[signature]		
Signature of inventor	Signature of inventor	Signature of inventor
06/30/2000		
Date	Date	Date

ON THE FLY GENERATION OF MULTIMEDIA CODE FOR IMAGE PROCESSING

Background of the Invention

Field of the Invention

The invention relates to the processing of multimedia data with processors that feature multimedia instruction enhanced instruction sets. More particularly, the invention relates to a method and apparatus for generating processor instruction sequences for image processing routines that use multimedia enhanced instructions.

Description of the Prior Art

In general, most programs that use image processing routines with multimedia instructions do not use a general-purpose compiler for these parts of the program. These programs typically use assembly routines to process such data. A resulting problem is that the assembly routines must be added to the code manually. This step requires high technical skill, is time demanding, and is prone to introduce errors into the code.

In addition, different type of processors, (for example, Intel's Pentium I w/MMX and Pentium II, Pentium III, Willamette, AMD's K-6 and AMD's K-7 aka. Athlon) each use different multimedia command sets. Examples of different multimedia command sets are MMX, SSE and 3DNow. Applications that use these multimedia command sets must have separate assembly routines that are specifically written for each processor type.

At runtime, the applications select the proper assembly routines based on the processor detected. To reduce the workload and increase the robustness of the code, these assembly routines are sometimes generated by a routine specific source code generator during program development.

One problem with this type of programming is that the applications must have redundant assembly routines which can process the same multimedia data, but which are written for the different types of processors. However, only one assembly routine is actually used at runtime. Because there are many generations of processors in existence, the size of applications that use multimedia instructions must grow to be compatible with all of these processors. In addition, as new processors are developed, all new routines must be coded for these applications so that they are compatible with the new processors. An application that is released prior to the release of a processor is incompatible

with the processor unless it is first patched/rebuilt with the new assembly routines.

It would be desirable to provide programs that use multimedia instructions which are smaller in size. It would be desirable to provide an approach that adapts such programs to future processors more easily

Summary of the Invention

In accordance with the invention, a method and apparatus for generating assembly routines for multimedia instruction enhanced data is shown and described.

An example of multimedia data that can be processed by multimedia instructions are the pixel blocks used in image processing. Most image processing routines operate on rectangular blocks of evenly sized data pieces (*e.g.* 16x16 pixel blocks of 8 bit video during MPEG motion compensation). The image processing code is described as a set of source blocks, destination blocks and data manipulations. Each block has a start address, a pitch (distance in bytes between two consecutive lines) and a data format. The full processing code includes width and height as additional parameters. All of these parameters can either be integer constants or arguments to the generated routine. All data operations are described on SIMD data types. A SIMD data type is a basic data

type (*e.g.* signed byte, signed word, or unsigned byte) and a number or repeats (*e.g.* 16 pixels for MPEG Macroblocks). The size of a block (source or destination) is always the size of its SIMD data type times its width in horizontal direction and the height in vertical direction.

In the presently preferred embodiment of the invention, an abstract image generator inside the application program produces an abstract routine representation of the code that operates on the multimedia data using SIMD operations. A directed acyclic graph is a typical example of a generic version. A translator then generates processor specific assembly code from the abstract respresentation.

Brief Description of the Drawings

FIG. 1 is a block diagram of a computer system that may be used to implement a method and apparatus embodying the invention for translating a multimedia routine from its abstract representation generated by an abstract routine generator inside the application's startup code into executable code using the code generator.

Description of the Preferred Embodiment

In Fig.1 the startup code 11 of the application program 13, further referred to as the abstract routine generator, generates an abstract representation 15 of the multimedia routine represented by a data flow graph. This graph is then translated by the code generator 17 into a machine specific sequence of instructions 19, typically including several SIMD multimedia instructions. The types of operations that can be present inside the data flow graph include add, sub, multiply, average, maximum, minimum, compare, and, or, xor, pack, unpack and merge operations. This list is not exhaustive as there are operations currently performed by MMX, SSE and 3DNow for example, which are not listed. If a specific command set does not support one of these operations, the CPU specific part of the code generator replaces it by a sequence of simpler instructions (e.g. the maximum instruction can be replaced by a pair of subtract and add instruction using saturation arithmetic).

The abstract routine generator generates an abstract representation of the code, commonly in the form of a directed acyclic graph during runtime. This allows the creation of multiple similar routines using a loop inside the image processing code 21 for linear arrays, or to generate routines on the fly depending on user interaction. *E.g.* the bi-directional MPEG 2 motion compensation can be implemented using a set of sixty-four different but very similar routines, that can be generated by a loop in the abstract image generator. Or an interactive paint

program can generate filters or pens in the form of abstract representations based on user input, and can use the routine generator to create efficient code sequences to perform the filtering or drawing operation. Examples of the data types processed by the code sequences include: SIMD input data, image input data and audio input data.

Examples of information provided by the graphs include the source blocks, the target blocks, the change in the block, color, stride, change in stride, display block, and spatial filtering.

The accuracy of the operation inside the graphs can be tailored to meet the requirements of the program. The abstract routine generator can increase its precision by increasing the level of arithmetics per pixel. For example, 7-bit processing can be stepped up to 8-bit, or 8-bit to 16-bit. *E.g.* motion compensation routines with different types of rounding precision can be generated by the abstract routine generator.

The abstract representation, in this case the graph 15, is then sent to the translator 17 where it is translated into optimized assembly code 19. The translator uses standard compiler techniques to translate the generic graph structure into a specific sequence of assembly instructions. As the description is very generic, there is no link to a specific processor architecture, and because it

is very simple it can be processed without requiring complex compiler techniques. This enables the translation to be executed during program startup without causing a significant delay. Also, the abstract generator and the translator do not have to be programmed in assembly. The CPU specific translator may reside in a dynamic link library and can therefore be replaced if the system processor is changed. This enables programs to use the multimedia instructions of a new processor, without the need to be changed.

Tables A-C provide sample code that generates an abstract representation for a motion compensation code that can be translated to an executable code sequence using the invention.

TABLE A

```
#ifndef MPEG2MOTIONCOMPENSATION_H
#define MPEG2MOTIONCOMPENSATION_H

#include "driver\softwarecinemaster\common\prelude.h"
#include "..\..\BlockVideoProcessor\BVPXMMXCodeConverter.h"

   //
   // Basic block motion compensation functions
   //
class MPEG2MotionCompensation
   {
   protected:
         //
         // Function prototype for a unidirectional motion compensation
routine
         //
      typedef void (__stdcall * CompensationCodeType)(BYTE * source1Base,
int sourceStride,
                                         BYTE * targetBase, short * deltaBase,
int deltaStride,
                                         int num);
```

```
        //
        // Function prototype for a bidirectional motion compensation
routine
        //
      typedef void (__stdcall * BiCompensationCodeType)(BYTE *
source1Base, BYTE * source2Base, int sourceStride,
                                   BYTE * targetBase, short * deltaBase,
int deltaStride,
                                   int num);


        //
        // Motion compensation routines for unidirectional prediction.
Each routine
        // handles one case.  The indices are
        //  - y-uv   : if it is luma data the index is 0 otherwise 1
        //  - delta  : error correction data is present (eg. the block
is not skipped)
        //  - halfy  : half pel prediction is to be performed in
vertical direction
        //  - halfx  : half pel prediction is to be performed in
horizontal direction
        //
      CompensationCodeType       compensation[2][2][2][2];            //
y-uv delta halfy halfx
      BVPCodeBlock           * compensationBlock[2][2][2][2];

        //
        // Motion compensation routines for bidirectional prediction.
Each routine
        // handles one case.  The indices contain the same parameters as
in the
        // unidirectional case, plus the half pel selectors for the
second source
        //
      BiCompensationCodeType     bicompensation[2][2][2][2][2][2];    //
y-uv delta half1y half1x half2y half2x
      BVPCodeBlock           * bicompensationBlock[2][2][2][2][2][2];
   public:
        //
        // Perform a unidirectional compensation
        //
      void MotionCompensation(BYTE * sourcep, int stride, BYTE * destp,
short * deltap, int dstride, int num, bool uv, bool delta, int halfx,
int halfy)
        {
        compensation[uv][delta][halfy][halfx](sourcep, stride, destp,
deltap, dstride, num);
        }

        //
        // Perform bidirectional compensation
        //
      void BiMotionCompensation(BYTE * source1p, BYTE * source2p, int
stride, BYTE * destp, short * deltap, int dstride, int num, bool uv,
bool delta, int half1x, int half1y, int half2x, int half2y)
```

```
        {
  bicompensation[uv][delta][half1y][half1x][half2y][half2x](source1p,
source2p, stride, destp, deltap, dstride, num);
        }

    MPEG2MotionCompensation(void);
    ~MPEG2MotionCompensation(void);
  };

#endif
```

TABLE B

```
#include "MPEG2MotionCompensation.h"

#include "..\..\BlockVideoProcessor\BVPXMMXCodeConverter.h"

  //
  // Create the dataflow to fetch a data element from a source block,
  // with or without half pel compensation in horizontal and/or
  // vertical direction.
  //
BVPDataSourceInstruction * BuildBlockMerge(BVPSourceBlock *
source1BlockA,
                                   BVPSourceBlock * source1BlockB,
                                   BVPSourceBlock * source1BlockC,
                                   BVPSourceBlock * source1BlockD,
                                   int halfx, int halfy)
  {
  if (halfy)
    {
    if (halfx)
      {
      //
      // Half pel prediction in h and v direction, the graph part
looks like this
      //
      //                    .--(LOAD source1BlockA)
      //                   /
      //           .--(AVG)
      //          /        \
      //         /          `--(LOAD source1BlockB)
      // <--(AVG)
      //         \          .--(LOAD source1BlockC)
      //          \        /
      //           `--(AVG)
      //                   \
      //                    `--(LOAD source1BlockD)
      //
      return new BVPDataOperation
            (
```

```
                    BVPDO_AVG,
                    new BVPDataOperation
                      (
                      BVPDO_AVG,
                      new BVPDataLoad(source1BlockA),
                      new BVPDataLoad(source1BlockB)
                      ),
                    new BVPDataOperation
                      (
                      BVPDO_AVG,
                      new BVPDataLoad(source1BlockC),
                      new BVPDataLoad(source1BlockD)
                      )
                    );
            }
          else
            {
            //
            // Half pel prediction in vertical direction
            //
            //          .--(LOAD source1BlockA)
            //         /
            // <--(AVG)
            //         \
            //          `--(LOAD source1BlockC)
            //
            return new BVPDataOperation
                   (
                   BVPDO_AVG,
                   new BVPDataLoad(source1BlockA),
                   new BVPDataLoad(source1BlockC)
                   );
            }
        }
      else
        {
        if (halfx)
            {
            //
            // Half pel prediction in horizontal direction
            //
            //          .--(LOAD source1BlockA)
            //         /
            // <--(AVG)
            //         \
            //          `--(LOAD source1BlockB)
            //
            return new BVPDataOperation
                   (
                   BVPDO_AVG,
                   new BVPDataLoad(source1BlockA),
                   new BVPDataLoad(source1BlockB)
                   );
            }
        else
            {
            //
```

```
        // Full pel prediction
        //
        // <--(LOAD source1BlockA)
        //
        return new BVPDataLoad(source1BlockA);
        }
      }
    }

MPEG2MotionCompensation::MPEG2MotionCompensation(void)
    {
    int yuv, delta, halfy, halfx, half1y, half1x, half2y, half2x;
    BVPBlockProcessor   * bvp;
    BVPCodeBlock      * code;

    BVPArgument * source1Base;
    BVPArgument * source2Base;
    BVPArgument * sourceStride;
    BVPArgument * targetBase;
    BVPArgument * deltaBase;
    BVPArgument * deltaStride;
    BVPArgument * height;

    BVPSourceBlock * source1BlockA;
    BVPSourceBlock * source1BlockB;
    BVPSourceBlock * source1BlockC;
    BVPSourceBlock * source1BlockD;
    BVPSourceBlock * source2BlockA;
    BVPSourceBlock * source2BlockB;
    BVPSourceBlock * source2BlockC;
    BVPSourceBlock * source2BlockD;

    BVPSourceBlock * deltaBlock;
    BVPTargetBlock * targetBlock;

    BVPDataSourceInstruction * postMC;
    BVPDataSourceInstruction * postCorrect;
    BVPDataSourceInstruction * deltaData;

    //
    // Build unidirectional motion compensation routines
    //
    for(yuv = 0; yuv<2; yuv++)
      {
      for(delta=0; delta<2; delta++)
        {
        for(halfy=0; halfy<2; halfy++)
          {
          for(halfx=0; halfx<2; halfx++)
            {
            bvp = new BVPBlockProcessor();

            bvp->AddArgument(height         = new BVPArgument(false));
            bvp->AddArgument(deltaStride    = new BVPArgument(false));
            bvp->AddArgument(deltaBase      = new BVPArgument(true));
            bvp->AddArgument(targetBase     = new BVPArgument(true));
            bvp->AddArgument(sourceStride   = new BVPArgument(false));
```

```
            bvp->AddArgument(source1Base     = new BVPArgument(true));

            //
            // Width is always sixteen pixels, so one vector of sixteen
unsigned eight bit elements,
            // height may vary, therefore it is an argument
            //
            bvp->SetDimension(1, height);

            //
            // Four potential source blocks, B is one pel to the right,
C one down and D right and down
            //
            bvp->AddSourceBlock(source1BlockA = new
BVPSourceBlock(source1Base,
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
            bvp->AddSourceBlock(source1BlockB = new
BVPSourceBlock(BVPPointer(source1Base, 1 + yuv),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
            bvp->AddSourceBlock(source1BlockC = new
BVPSourceBlock(BVPPointer(source1Base, sourceStride, 1, 0),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
            bvp->AddSourceBlock(source1BlockD = new
BVPSourceBlock(BVPPointer(source1Base, sourceStride, 1, 1 + yuv),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));

            //
            // If we have error correction data, we need this source
block as well
            //
            if (delta)
              bvp->AddSourceBlock(deltaBlock    = new
BVPSourceBlock(deltaBase, deltaStride, BVPDataFormat(BVPDT_S16, 16),
0x10000));

            //
            // The target block to write the data into
            //
            bvp->AddTargetBlock(targetBlock   = new
BVPTargetBlock(targetBase, sourceStride, BVPDataFormat(BVPDT_U8,  16),
0x10000));

            //
            // Load a source block base on the half pel settings
            //
            bvp->AddInstruction(postMC = BuildBlockMerge(source1BlockA,
source1BlockB, source1BlockC, source1BlockD, halfx, halfy));

            if (delta)
               {
               deltaData = new BVPDataLoad(deltaBlock);

               if (yuv)
                  {
                  //
                  // It is chroma data and we have error correction data.
The u and v
```

```
                    // parts have to be interleaved, therefore we need the
merge instruction
                    //
                    //                         .--(CONV S16)<--postMC
                    //                        /
                    // <--(CONV U8)<--(ADD)
                    //                        \                .--(SPLIT H)<-.
                    //                         \              /               \
                    //                          `--(MERGE OE)
>--(LOAD delta)
                    //                                        \               /
                    //                                         `--(SPLIT T)<-¥
                    //
                    bvp->AddInstruction
                       (
                       postCorrect =
                       new BVPDataConvert
                          (
                          BVPDT_U8,
                          new BVPDataOperation
                             (
                             BVPDO_ADD,
                             new BVPDataConvert
                                (
                                BVPDT_S16,
                                postMC
                                ),
                             new BVPDataMerge
                                (
                                BVPDM_ODDEVEN,
                                new BVPDataSplit
                                   (
                                   BVPDS_HEAD,
                                   deltaData
                                   ),
                                new BVPDataSplit
                                   (
                                   BVPDS_TAIL,
                                   deltaData
                                   )
                                )
                             )
                          )
                       );
                    }
                  else
                    {
                    //
                    // It is luma data with error correction
                    //
                    //                         .--(CONV S16)<--postMC
                    //                        /
                    // <--(CONV U8)<--(ADD)
                    //                        \
                    //                         `--(LOAD delta)
                    //
                    bvp->AddInstruction
```

```
                    (
                    postCorrect =
                    new BVPDataConvert
                      (
                      BVPDT_U8,
                      new BVPDataOperation
                        (
                        BVPDO_ADD,
                        new BVPDataConvert
                          (
                          BVPDT_S16,
                          postMC
                          ),
                        deltaData
                        )
                      )
                    );
                  }

                //
                // Store into the target block
                //
                // (STORE targetBlock)<--...
                //
                bvp->AddInstruction
                  (
                  new BVPDataStore
                    (
                    targetBlock,
                    postCorrect
                    )
                  );
                }
              else
                {
                //
                // No error correction data, so store motion result into
target block
                //
                // (STORE targetBlock)<--...
                //
                bvp->AddInstruction
                  (
                  new BVPDataStore
                    (
                    targetBlock,
                    postMC
                    )
                  );
                }

              BVPXMMXCodeConverter conv;

              //
              // Convert graph into machine language
              //
```

```
                    compensationBlock[yuv][delta][halfy][halfx] = code =
conv.Convert(bvp);

                    //
                    // Get function entry pointer
                    //
                    compensation[yuv][delta][halfy][halfx] =
(CompensationCodeType)(code->GetCodeAddress());

                    //
                    // delete graph
                    //
                    delete bvp;
                    }
                }
            }
        }

    //
    // build motion compensation routines for bidirectional prediction
    //
    for(yuv = 0; yuv<2; yuv++)
        {
        for(delta=0; delta<2; delta++)
            {
            for(half1y=0; half1y<2; half1y++)
                {
                for(half1x=0; half1x<2; half1x++)
                    {
                    for(half2y=0; half2y<2; half2y++)
                        {
                        for(half2x=0; half2x<2; half2x++)
                            {
                            bvp = new BVPBlockProcessor();

                            bvp->AddArgument(height        = new
BVPArgument(false));
                            bvp->AddArgument(deltaStride   = new
BVPArgument(false));
                            bvp->AddArgument(deltaBase     = new
BVPArgument(true));
                            bvp->AddArgument(targetBase    = new
BVPArgument(true));
                            bvp->AddArgument(sourceStride  = new
BVPArgument(false));
                            bvp->AddArgument(source2Base   = new
BVPArgument(true));
                            bvp->AddArgument(source1Base   = new
BVPArgument(true));

                            bvp->SetDimension(1, height);

                            //
                            // We now have two source blocks, so we need eight
blocks for the half pel
                            // prediction
                            //
```

```
                    bvp->AddSourceBlock(source1BlockA = new
BVPSourceBlock(source1Base,
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
                    bvp->AddSourceBlock(source1BlockB = new
BVPSourceBlock(BVPPointer(source1Base, 1 + yuv),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
                    bvp->AddSourceBlock(source1BlockC = new
BVPSourceBlock(BVPPointer(source1Base, sourceStride, 1, 0),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
                    bvp->AddSourceBlock(source1BlockD = new
BVPSourceBlock(BVPPointer(source1Base, sourceStride, 1, 1 + yuv),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
                    bvp->AddSourceBlock(source2BlockA = new
BVPSourceBlock(source2Base,
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
                    bvp->AddSourceBlock(source2BlockB = new
BVPSourceBlock(BVPPointer(source2Base, 1 + yuv),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
                    bvp->AddSourceBlock(source2BlockC = new
BVPSourceBlock(BVPPointer(source2Base, sourceStride, 1, 0),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));
                    bvp->AddSourceBlock(source2BlockD = new
BVPSourceBlock(BVPPointer(source2Base, sourceStride, 1, 1 + yuv),
sourceStride,  BVPDataFormat(BVPDT_U8, 16), 0x10000));

                    if (delta)
                       bvp->AddSourceBlock(deltaBlock   = new
BVPSourceBlock(deltaBase, deltaStride, BVPDataFormat(BVPDT_S16, 16),
0x10000));

                    bvp->AddTargetBlock(targetBlock  = new
BVPTargetBlock(targetBase, sourceStride, BVPDataFormat(BVPDT_U8,  16),
0x10000));

                    //
                    // Build bidirectional prediction from two
unidirectional predictions
                    //
                    //          .--BuildBlockMerge(source1Block*)
                    //         /
                    // <--(AVG)
                    //         \
                    //          `--BuildBlockMerge(source2Block*)
                    //
                    bvp->AddInstruction
                       (
                       postMC =
                       new BVPDataOperation
                          (
                          BVPDO_AVG,
                          BuildBlockMerge(source1BlockA, source1BlockB,
source1BlockC, source1BlockD, half1x, half1y),
                          BuildBlockMerge(source2BlockA, source2BlockB,
source2BlockC, source2BlockD, half2x, half2y)
                          )
                       );
```

```
//
// Apply error correction, see unidirectional case
//
if (delta)
   {
   deltaData = new BVPDataLoad(deltaBlock);

   if (yuv)
      {
      bvp->AddInstruction
         (
         postCorrect =
         new BVPDataConvert
            (
            BVPDT_U8,
            new BVPDataOperation
               (
               BVPDO_ADD,
               new BVPDataConvert
                  (
                  BVPDT_S16,
                  postMC
                  ),
               new BVPDataMerge
                  (
                  BVPDM_ODDEVEN,
                  new BVPDataSplit
                     (
                     BVPDS_HEAD,
                     deltaData
                     ),
                  new BVPDataSplit
                     (
                     BVPDS_TAIL,
                     deltaData
                     )
                  )
               )
            )
         );
      }
   else
      {
      bvp->AddInstruction
         (
         postCorrect =
         new BVPDataConvert
            (
            BVPDT_U8,
            new BVPDataOperation
               (
               BVPDO_ADD,
               new BVPDataConvert
                  (
                  BVPDT_S16,
                  postMC
                  ),
```

```
                                        deltaData
                                        )
                                    )
                                );
                            }

                        bvp->AddInstruction
                            (
                            new BVPDataStore
                                (
                                targetBlock,
                                postCorrect
                                )
                            );
                        }
                    else
                        {
                        bvp->AddInstruction
                            (
                            new BVPDataStore
                                (
                                targetBlock,
                                postMC
                                )
                            );
                        }

                    BVPXMMXCodeConverter conv;

                    //
                    // Translate routines
                    //

    bicompensationBlock[yuv][delta][half1y][half1x][half2y][half2x] =
code = conv.Convert(bvp);


    bicompensation[yuv][delta][half1y][half1x][half2y][half2x] =
(BiCompensationCodeType)(code->GetCodeAddress());

                    delete bvp;
                    }
                }
            }
        }
    }
  }
}

MPEG2MotionCompensation::~MPEG2MotionCompensation(void)
    {
    int yuv, delta, halfy, halfx, half1y, half1x, half2y, half2x;

    //
    // free all motion compensation routines
    //
    for(yuv = 0; yuv<2; yuv++)
```

```
        {
        for(delta=0; delta<2; delta++)
          {
          for(halfy=0; halfy<2; halfy++)
            {
            for(halfx=0; halfx<2; halfx++)
              {
              delete compensationBlock[yuv][delta][halfy][halfx];
              }
            }
          }
        }
      for(yuv = 0; yuv<2; yuv++)
        {
        for(delta=0; delta<2; delta++)
          {
          for(half1y=0; half1y<2; half1y++)
            {
            for(half1x=0; half1x<2; half1x++)
              {
              for(half2y=0; half2y<2; half2y++)
                {
                for(half2x=0; half2x<2; half2x++)
                  {
                  delete
bicompensationBlock[yuv][delta][half1y][half1x][half2y][half2x];
                  }
                }
              }
            }
          }
        }
      }
```

TABLE C

```
#ifndef BVPGENERIC_H
#define BVPGENERIC_H

#include "BVPList.h"

  //
  // Argument descriptor.  An argument can be either a pointer or an
integer used
  // as a stride, offset or width/height value.
  //
class BVPArgument
  {
  public:
    bool pointer;
    int  index;
```

```
      BVPArgument(bool pointer_)
         : pointer(pointer_), index(0) {}
   };

   //
   // Description of an integer value used as a stride or offset.  An
integer value
   // can be either an argument or a constant
   //
class BVPInteger
   {
   public:
      int         value;
      BVPArgument * arg;

      BVPInteger(void)
         : value(0), arg(NULL) {}
      BVPInteger(int value_)
         : value(value_), arg(NULL) {}
      BVPInteger(unsigned value_)
         : value((int)value_), arg(NULL) {}
      BVPInteger(BVPArgument * arg_)
         : value(0), arg(arg_) {}

      bool operator== (BVPInteger i2)
         {
         return arg ? (i2.arg == arg) : (i2.value == value);
         }
   };

   //
   // Description of a memory pointer used as a base for source and
target blocks.
   // A pointer can be a combination of an pointer base, a constant
offset and
   // a variable index with scaling
   //
class BVPPointer
   {
   public:
      BVPArgument * base;
      BVPArgument * index;
      int         offset;
      int         scale;

      BVPPointer(BVPArgument * base_)
         : base(base_), index(NULL), offset(0), scale(0) {}

      BVPPointer(BVPPointer base_, int offset_)
         : base(base_.base), index(NULL), offset(offset_), scale(0) {}

      BVPPointer(BVPPointer base_, BVPInteger index_, int scale_, int
offset_)
         : base(base_.base), index(index_.arg), offset(offset_),
scale(scale_) {}
   };
```

```
//
// Base data formats for scalar types
//
enum BVPBaseDataFormat
{
BVPDT_U8,    // Unsigned 8 bits
BVPDT_U16,     // Unsigned 16 bits
BVPDT_U32,     // Unsigned 32 bits
BVPDT_S8,    // Signed 8 bits
BVPDT_S16,     // Signed 16 bits
BVPDT_S32    // Signed 32 bits
};

//
// Data forma descriptor for scalar and vector (multimedia simd)
types
// Each data type is a combination of a base type and a vector size.
// Scalar types are represented by a vector size of one.
//
class BVPDataFormat
{
public:
   BVPBaseDataFormat   format;
   int             num;

   BVPDataFormat(BVPBaseDataFormat _format, int _num = 1)
      : format(_format), num(_num) {}

   BVPDataFormat(void)
      : format(BVPDT_U8), num(0) {}

   BVPDataFormat(BVPDataFormat & f)
      : format(f.format), num(f.num) {}

   BVPDataFormat operator* (int times)
      {return BVPDataFormat(format, num * times);}

   BVPDataFormat operator/ (int times)
      {return BVPDataFormat(format, num / times);}

   int BitsPerElement(void) {static const int sz[] = {8, 16, 32, 8,
16, 32}; return sz[format];}
   int BitsPerChunk(void) {return BitsPerElement() * num;}
};

//
// Operation codes for binary data operations that have the
// same operand type for both sources and the destination
//
enum BVPDataOperationCode
{
BVPDO_ADD,              // add with wraparound
BVPDO_ADD_SATURATED,      // add with saturation
BVPDO_SUB,              // subtract with wraparound
BVPDO_SUB_SATURATED,      // subtract with saturation
BVPDO_MAX,              // maximum
```

```
    BVPDO_MIN,              // minimum
    BVPDO_AVG,              // average (includes rounding towards nearest)
    BVPDO_EQU,              // equal
    BVPDO_OR,             // binary or
    BVPDO_XOR,              // binary exclusive or
    BVPDO_AND,              // binary and
    BVPDO_ANDNOT,           // binary and not
    BVPDO_MULL,             // multiply keep lower half
    BVPDO_MULH              // multiply keep upper half
    };

    //
    // Operations that extract a part of a data element
    //
  enum BVPDataSplitCode
    {
    BVPDS_HEAD,             // extract first half
    BVPDS_TAIL,             // extract second half
    BVPDS_ODD,              // extract odd elements
    BVPDS_EVEN              // extract even elements
    };

    //
    // Operations that combine to data elements
    //
  enum BVPDataMergeCode
    {
    BVPDM_UPPERLOWER,         // chain first and second operands
    BVPDM_ODDEVEN           // interleave first and second operands
    };

    //
    // Node types in the data flow graph
    //
  enum BVPInstructionType
    {
    BVPIT_LOAD,             // load an element from a source block
    BVPIT_STORE,            // store an element into a source block
    BVPIT_CONSTANT,         // load a constant value
    BVPIT_SPLIT,            // split an element
    BVPIT_MERGE,            // merge two elements
    BVPIT_CONVERT,          // perform a data conversion
    BVPIT_OPERATION         // simple binary data operation
    };

    //
    // Descriptor of a data block.  Contains a base pointer, a
  stride(pitch), a
    // format and an incrementor in vertical direction.  The vertical
  block position
    // can be incremented by a fraction or a multiple of the given pitch.
    //
  class BVPBlock
    {
    public:
      BVPPointer    base;
      BVPInteger    pitch;
```

```
    BVPDataFormat  format;
    int          yscale;
    int          index;

    BVPBlock(BVPPointer _base, BVPInteger _pitch, BVPDataFormat
_format, int _yscale)
        : base(_base), pitch(_pitch), format(_format), yscale(_yscale)
{}
  };

  //
  // Descriptor of a source block
  //
class BVPSourceBlock : public BVPBlock
  {
  public:
    BVPSourceBlock(BVPPointer base, BVPInteger pitch, BVPDataFormat
format, int yscale)
        : BVPBlock(base, pitch, format, yscale) {}
  };

  //
  // Descriptor of a target block
  //
class BVPTargetBlock : public BVPBlock
  {
  public:
    BVPTargetBlock(BVPPointer base, BVPInteger pitch, BVPDataFormat
format, int yscale)
        : BVPBlock(base, pitch, format, yscale) {}
  };

class BVPDataSource;
class BVPDataDrain;
class BVPDataInstruction;

  //
  // Source connection element of a node in the data flow graph.  Each
node in
  // the graph contains one or none source connection.  A source
connection is
  // the output of a node in the graph. Each source connection can be
connected
  // to any number of drain connections in other nodes of the flow
graph.  The
  // source is the output side of a node.
  //
class BVPDataSource
  {
  public:
    BVPDataFormat             format;
    BVPList<BVPDataDrain *>    drain;

    BVPDataSource(BVPDataFormat _format) : format(_format) {}

    virtual void AddInstructions(BVPList<BVPDataInstruction *> &
instructions) {}
```

```
    virtual BVPDataInstruction * ToInstruction(void) {return NULL;}
  };

  //
  // Drain connection element of a node in the data flow graph.  Each
node
  // can have none, one or two drain connections (but only one drain
object
  // to represent both).  Each drain connects to exactly one source on
the
  // target side.  As eachnode can have only two inputs, each drain is
connected
  // (through the node) with two sources.  The drain is the input side
of a
  // node.
  //
class BVPDataDrain
  {
  public:
    BVPDataSource         * source1;
    BVPDataSource         *  source2;

    BVPDataDrain(BVPDataSource * source1_, BVPDataSource * source2_ =
NULL)
        : source1(source1_), source2(source2_) {}

    virtual BVPDataInstruction * ToInstruction(void) {return NULL;}
  };

  //
  // Each node in the graph represents one abstract instruction.  It
has an
  // instruction type that describes the operation of the node.
  //
class BVPDataInstruction
  {
  public:
    BVPInstructionType type;
    int                index;

    BVPDataInstruction(BVPInstructionType type_)
      : type(type_), index(-1) {}

    virtual ~BVPDataInstruction(void) {}

    virtual void AddInstructions(BVPList<BVPDataInstruction *> &
instructions);
    virtual void GetOperationBits(int & minBits, int & maxBits);

    virtual BVPDataFormat GetInputFormat(void) = 0;
    virtual BVPDataFormat GetOutputFormat(void) = 0;

    virtual BVPDataSource * ToSource(void) {return NULL;}
    virtual BVPDataDrain * ToDrain(void) {return NULL;}
  };

  //
```

```
// Node that is a data source
//
class BVPDataSourceInstruction : public BVPDataInstruction, public
BVPDataSource
  {
  public:
    BVPDataSourceInstruction(BVPInstructionType type_, BVPDataFormat
format_)
      : BVPDataInstruction(type_), BVPDataSource(format_) {}

    void GetOperationBits(int & minBits, int & maxBits);

    BVPDataFormat GetOutputFormat(void) {return format;}
    BVPDataFormat GetInputFormat(void) {return format;}

    BVPDataInstruction * ToInstruction(void) {return this;}
    BVPDataSource * ToSource(void) {return this;}
  };

  //
  // Node that is a data source and has one or two sources connected to
its drain
  //
class BVPDataSourceDrainInstruction : public BVPDataSourceInstruction,
public BVPDataDrain
  {
  public:
    BVPDataSourceDrainInstruction(BVPInstructionType type_,
BVPDataFormat format_, BVPDataSource * source1_)
      : BVPDataSourceInstruction(type_, format_),
BVPDataDrain(source1_)
      {source1->drain.Insert(this);}
    BVPDataSourceDrainInstruction(BVPInstructionType type_,
BVPDataFormat format_, BVPDataSource * source1_, BVPDataSource *
source2_)
      : BVPDataSourceInstruction(type_, format_),
BVPDataDrain(source1_, source2_)
      {source1->drain.Insert(this);source2->drain.Insert(this);}
  };

  //
  // Instruction to load data from a source block
  //
class BVPDataLoad : public BVPDataSourceInstruction
  {
  public:
    BVPSourceBlock   * block;
    int              offset;

    BVPDataLoad(BVPSourceBlock * block_, int offset_ = 0)
      : BVPDataSourceInstruction(BVPIT_LOAD, block_->format),
block(block_), offset(offset_) {}

    void AddInstructions(BVPList<BVPDataInstruction *> & instructions);
  };

  //
```

```
    // Instruction to store data into a target block
    //
class BVPDataStore : public BVPDataInstruction, public BVPDataDrain
    {
    public:
        BVPTargetBlock   * block;

        BVPDataStore(BVPTargetBlock * block_, BVPDataSource * source)
            : BVPDataInstruction(BVPIT_STORE), BVPDataDrain(source),
block(block_)
            {source->drain.Insert(this);}

        void AddInstructions(BVPList<BVPDataInstruction *> & instructions);

        BVPDataFormat GetOutputFormat(void) {return source1->format;}
        BVPDataFormat GetInputFormat(void) {return source1->format;}

        BVPDataInstruction * ToInstruction(void) {return this;}
        BVPDataDrain * ToDrain(void) {return this;}
    };

    //
    // Instruction to load a constant
    //
class BVPDataConstant : public BVPDataSourceInstruction
    {
    public:
        int value;

        BVPDataConstant(BVPDataFormat format, int value_)
            : BVPDataSourceInstruction(BVPIT_CONSTANT, format),
value(value_) {}
    };

    //
    // Instruction to split a data element
    //
class BVPDataSplit : public BVPDataSourceDrainInstruction
    {
    public:
        BVPDataSplitCode code;

        BVPDataSplit(BVPDataSplitCode code_, BVPDataSource * source)
            : BVPDataSourceDrainInstruction(BVPIT_SPLIT, source->format / 2,
source), code(code_) {}

        void AddInstructions(BVPList<BVPDataInstruction *> & instructions);

        BVPDataDrain * ToDrain(void) {return this;}

        BVPDataFormat GetInputFormat(void) {return source1->format;}
    };

    //
    // Instruction to merge two data elements
    //
class BVPDataMerge : public BVPDataSourceDrainInstruction
```

```
    {
    public:
        BVPDataMergeCode code;

        BVPDataMerge(BVPDataMergeCode code_, BVPDataSource * source1_,
BVPDataSource * source2_)
            : BVPDataSourceDrainInstruction(BVPIT_MERGE, source1_->format *
2, source1_, source2_),
              code(code_) {}

        void AddInstructions(BVPList<BVPDataInstruction *> & instructions);

        BVPDataDrain * ToDrain(void) {return this;}

        BVPDataFormat GetInputFormat(void) {return source1->format;}
    };

    //
    // Instruction to convert the basic vector elements of an data
element into
    // a different format (eg. from signed 16 bit to unsigned 8 bits).
    //
class BVPDataConvert : public BVPDataSourceDrainInstruction
    {
    public:
        BVPDataConvert(BVPBaseDataFormat target, BVPDataSource * source)
            : BVPDataSourceDrainInstruction(BVPIT_CONVERT,
BVPDataFormat(target, source->format.num), source) {}

        void AddInstructions(BVPList<BVPDataInstruction *> & instructions);

        BVPDataDrain * ToDrain(void) {return this;}

        BVPDataFormat GetInputFormat(void) {return source1->format;}
    };

    //
    // Basic data manipulation operation from two sources to one drain.
    //
class BVPDataOperation : public BVPDataSourceDrainInstruction
    {
    public:
        BVPDataOperationCode  code;

        BVPDataOperation(BVPDataOperationCode code_, BVPDataSource *
source1_, BVPDataSource * source2_)
            : BVPDataSourceDrainInstruction(BVPIT_OPERATION, source1_-
>format, source1_, source2_), code(code_) {}

        void AddInstructions(BVPList<BVPDataInstruction *> & instructions);

        BVPDataDrain * ToDrain(void) {return this;}
    };

    //
    // Descriptor for one image block processing routine.  It contains
the arguments, the
```

```
  // size and the dataflow graph.  On destruction of the block
processor all argument,
  // blocks and instructions are also deleted.
  //
class BVPBlockProcessor
  {
  public:
    BVPInteger  width;
    BVPInteger  height;

    BVPList<BVPBlock *> blocks;
    BVPList<BVPDataInstruction *> instructions;
    BVPList<BVPArgument *> args;

    BVPBlockProcessor(void)
      {
      }

    ~BVPBlockProcessor(void);

      //
      // Add an argument to the list of arguments.  Please note that
the arguments
      // are added in the reverse order of the c-calling convention.
      //
    void AddArgument(BVPArgument * arg)
      {
      arg->index = args.Num();
      args.Insert(arg);
      }

      //
      // Set the dimension of the operation rectangle.  The width and
height can
      // either be constants or arguments to the routine.
      //
    void SetDimension(BVPInteger width, BVPInteger height)
      {
      this->width = width;
      this->height = height;
      }

      //
      // Add a source block to the processing
      //
    void AddSourceBlock(BVPSourceBlock * block)
      {
      block->index = blocks.Num();
      blocks.Insert(block);
      }

      //
      // Add a target block to the processing
      //
    void AddTargetBlock(BVPTargetBlock * block)
      {
      block->index = blocks.Num();
```

```
        blocks.Insert(block);
        }

        //
        // Add an instruction to the dataflow graph.  All referenced
instructions
        // will also be added to the graph if they are not yet part of
it.
        //
      void AddInstruction(BVPDataInstruction * ins)
        {
        ins->AddInstructions(instructions);
        }

      void GetOperationBits(int & minBits, int & maxBits);
  };

#endif
```

Although the invention is described herein with reference to the preferred embodiment, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the claims included below.

Claims

1. An apparatus for generating computer assembly code, comprising:

an abstract routine generator for receiving a data stream comprising a multimedia routine and for outputting a generic abstract representation thereof; and

a translator for said abstract routine generator for receiving said abstract representation and for outputting processor specific code for processing multimedia input data.

2. The apparatus of Claim 1, where in said abstract routine generator builds an abstract routine during runtime.

3. The apparatus of Claim 1, wherein said abstract routine generator builds an abstract routine in the form of a graph.

4. The apparatus of Claim 1 wherein said multimedia data comprise SIMD input data.

5. The apparatus of Claim 1, wherein said multimedia data comprise image input data.

6. The apparatus of Claim 1, wherein said multimedia data comprise audio input data.

7. The apparatus of Claim 3, wherein said graph is input to said translator.

8. The apparatus of Claim 3, wherein the output of said translator is in assembly code.

9. The apparatus of Claim 1, wherein said translator's configuration can be changed by use of a dynamic library link.

10. The apparatus of Claim 1, wherein said processor-specific code performs any of the operations of add, sub, multiply, average, maximum, minimum, compare, and, or, xor, pack, unpack, and merge on said input data.

11. The apparatus of Claim 3, wherein said graph is a function of any of source block, target block, change in the block, color, stride, change in stride, display block, and spatial filtering.

12. A method for generating assembly code, comprising:

providing an abstract routine generator for generating a generic abstract representation of an input stream, and input comprising multimedia a routine; and

providing a translator for receiving said abstract representation from said abstract routine generator and for outputting processor-specific code for processing multimedia input data.

13. The method of Claim 12, wherein said abstract routine generator builds the abstract routine during runtime.

14. The method of Claim 13, wherein said abstract routine is a graph.

15. The method of Claim 12, wherein said multimedia input data comprise SIMD data.

16. The method of Claim 12, said multimedia input data comprise image data.

17. The method of Claim 12, wherein said multimedia input data comprise audio data.

18. The method of claim 14, wherein said graph is input to said translator.

19. The method of claim 12, wherein the output of said translator is assembly code.

20. The method of Claim 12, wherein said processor-specific code performs any of the operations of add, sub, multiply, average, maximum, minimum, compare, and, or, xor, pack, unpack, and merge on said multimedia input data.

21. The method of Claim 14, wherein said graph is a function of any of source block, target block, change in the block, color, stride, change in stride, display block, and spatial filtering.

22. The method of Claim 12, wherein said translator can be changed by use of a dynamic library link.

Abstract

A method and apparatus for processing multimedia instruction enhanced data by the use of an abstract routine generator and a translator. The abstract routine generator takes the multimedia instruction enhanced data and generates abstract routines to compile the multimedia instruction enhanced data. The output of the abstract generator is an abstract representation of the multimedia instruction enhanced data. The translator then takes the abstract representation and produces code for processing.

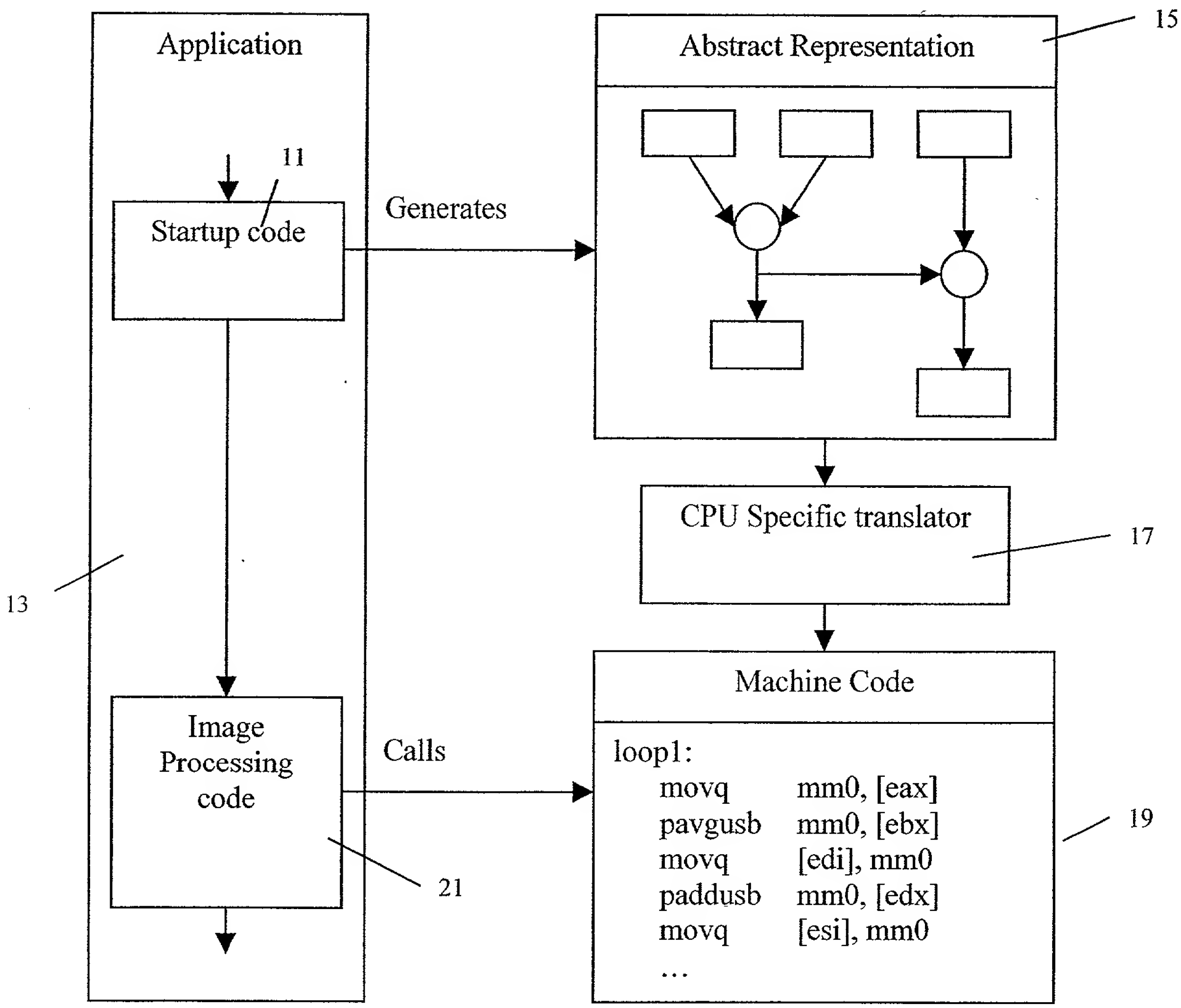
Application
11
Startup code
Generates
Abstract Representation
15
CPU Specific translator
17
13
Image Processing code
Calls
Machine Code
loop1:
movq mm0, [eax]
pavgusb mm0, [ebx]
movq [edi], mm0
paddusb mm0, [edx]
movq [esi], mm0
...
19
21

Fig. 1

Attorney Docket No. RAVI0009

DECLARATION FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address, and citizenship are as stated below next to my name;

I believe I am the original, first, and sole inventor (if only one name is listed below) or an original, first, and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

ON THE FLY GENERATION OF MULTIMEDIA CODE FOR IMAGE PROCESSING

the specification of which (check one) X is attached hereto, or ___ was filed on ____________________ as Application Serial No. ____________________ and was amended on _______ (if applicable).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56(a).

==

I hereby claim foreign priority benefits under Title 35, United Sates Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)	Priority Claimed Yes	Priority Claimed No
______________________________	____	____
Number Country Day/Month/Year Filed		
______________________________	____	____
Number Country Day/Month/Year Filed		

==

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

MICHAEL A. GLENN, Reg. No. 30,176
DONALD M. HENDRICKS, Reg. No. 40,355
KIRK D. WONG, Reg. NO. 43,284
EARLE W. JENNINGS, Reg. No. 44,804
CHRISTOPHER PEIL, Reg. No. 45,005

SEND CORRESPONDENCE TO:

MICHAEL A. GLENN, 3475 Edison Way, Suite L, Menlo Park, CA 94025

==

Attorney Docket No. RAVI0009

==

I hereby claim the benefit under Title 35, United States code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Application Ser. No.	Filing Date	Status: Patented, Pending, Abandoned

==

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole or first inventor: ULRICH SIGMUND

Inventor's signature [signature] 06/30/2000
Date

Residence Viktoriastr. 6, 76133 Karlsruhe, Germany

Post Office Address Same

Citizenship German